## Bike Sharing Dataset

#### Import pandas, numpy, seaborn, matplotlib.pyplot packages

In [1]:

**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**%**matplotlib inline

**import** seaborn **as** sns

**from** warnings **import** filterwarnings

filterwarnings('ignore')

#### Importing Dataset

In [7]:

df **=** pd.read\_csv('Datasets/hour.csv')

df.head()

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **instant** | **dteday** | **season** | **yr** | **mnth** | **hr** | **holiday** | **weekday** | **workingday** | **weathersit** | **temp** | **atemp** | **hum** | **windspeed** | **casual** | **registered** | **cnt** |
| **0** | 1 | 2011-01-01 | 1 | 0 | 1 | 0 | 0 | 6 | 0 | 1 | 0.24 | 0.2879 | 0.81 | 0.0 | 3 | 13 | 16 |
| **1** | 2 | 2011-01-01 | 1 | 0 | 1 | 1 | 0 | 6 | 0 | 1 | 0.22 | 0.2727 | 0.80 | 0.0 | 8 | 32 | 40 |
| **2** | 3 | 2011-01-01 | 1 | 0 | 1 | 2 | 0 | 6 | 0 | 1 | 0.22 | 0.2727 | 0.80 | 0.0 | 5 | 27 | 32 |
| **3** | 4 | 2011-01-01 | 1 | 0 | 1 | 3 | 0 | 6 | 0 | 1 | 0.24 | 0.2879 | 0.75 | 0.0 | 3 | 10 | 13 |
| **4** | 5 | 2011-01-01 | 1 | 0 | 1 | 4 | 0 | 6 | 0 | 1 | 0.24 | 0.2879 | 0.75 | 0.0 | 0 | 1 | 1 |

Out[7]:

**It is a Regression Problem - the Dependent variable is cnt(ie, count of total rental bikes)**

* **Shape of Dataset**

In [3]:

df.shape

Out[3]:

(17379, 17)

* **The dataset has total 17379 rows & 17 attributes**
* **Checking Information of Dataset**

In [4]:

df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 17379 entries, 0 to 17378  
Data columns (total 17 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 instant 17379 non-null int64   
 1 dteday 17379 non-null object   
 2 season 17379 non-null int64   
 3 yr 17379 non-null int64   
 4 mnth 17379 non-null int64   
 5 hr 17379 non-null int64   
 6 holiday 17379 non-null int64   
 7 weekday 17379 non-null int64   
 8 workingday 17379 non-null int64   
 9 weathersit 17379 non-null int64   
 10 temp 17379 non-null float64  
 11 atemp 17379 non-null float64  
 12 hum 17379 non-null float64  
 13 windspeed 17379 non-null float64  
 14 casual 17379 non-null int64   
 15 registered 17379 non-null int64   
 16 cnt 17379 non-null int64   
dtypes: float64(4), int64(12), object(1)  
memory usage: 2.3+ MB

* Dataset has 4 Float columns, 12 integer columns and 1 object (string) Columns

## Data preprocessing

Pre-processing techniques include:

* 1.Handling Missing Data
* 2.Removing Outliers
* 3.Encoding Categorical Text Variables
* 4.Feature Scaling

#### Check Null Values

In [8]:

df.isnull().sum()

Out[8]:

instant 0  
dteday 0  
season 0  
yr 0  
mnth 0  
hr 0  
holiday 0  
weekday 0  
workingday 0  
weathersit 0  
temp 0  
atemp 0  
hum 0  
windspeed 0  
casual 0  
registered 0  
cnt 0  
dtype: int64

* There is no Null Values

#### Now droping irrelevent columns

In [9]:

df **=** df.drop(['instant','dteday'], axis**=**1)

df.head()

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **season** | **yr** | **mnth** | **hr** | **holiday** | **weekday** | **workingday** | **weathersit** | **temp** | **atemp** | **hum** | **windspeed** | **casual** | **registered** | **cnt** |
| **0** | 1 | 0 | 1 | 0 | 0 | 6 | 0 | 1 | 0.24 | 0.2879 | 0.81 | 0.0 | 3 | 13 | 16 |
| **1** | 1 | 0 | 1 | 1 | 0 | 6 | 0 | 1 | 0.22 | 0.2727 | 0.80 | 0.0 | 8 | 32 | 40 |
| **2** | 1 | 0 | 1 | 2 | 0 | 6 | 0 | 1 | 0.22 | 0.2727 | 0.80 | 0.0 | 5 | 27 | 32 |
| **3** | 1 | 0 | 1 | 3 | 0 | 6 | 0 | 1 | 0.24 | 0.2879 | 0.75 | 0.0 | 3 | 10 | 13 |
| **4** | 1 | 0 | 1 | 4 | 0 | 6 | 0 | 1 | 0.24 | 0.2879 | 0.75 | 0.0 | 0 | 1 | 1 |

Out[9]:

* There is No Object variable so, no need for label encoding/one hot encoding

### Spliting Data into x and y

In [10]:

df.head()

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **season** | **yr** | **mnth** | **hr** | **holiday** | **weekday** | **workingday** | **weathersit** | **temp** | **atemp** | **hum** | **windspeed** | **casual** | **registered** | **cnt** |
| **0** | 1 | 0 | 1 | 0 | 0 | 6 | 0 | 1 | 0.24 | 0.2879 | 0.81 | 0.0 | 3 | 13 | 16 |
| **1** | 1 | 0 | 1 | 1 | 0 | 6 | 0 | 1 | 0.22 | 0.2727 | 0.80 | 0.0 | 8 | 32 | 40 |
| **2** | 1 | 0 | 1 | 2 | 0 | 6 | 0 | 1 | 0.22 | 0.2727 | 0.80 | 0.0 | 5 | 27 | 32 |
| **3** | 1 | 0 | 1 | 3 | 0 | 6 | 0 | 1 | 0.24 | 0.2879 | 0.75 | 0.0 | 3 | 10 | 13 |
| **4** | 1 | 0 | 1 | 4 | 0 | 6 | 0 | 1 | 0.24 | 0.2879 | 0.75 | 0.0 | 0 | 1 | 1 |

Out[10]:

In [11]:

x **=** df.drop(['cnt'], axis**=**1)

x.head()

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **season** | **yr** | **mnth** | **hr** | **holiday** | **weekday** | **workingday** | **weathersit** | **temp** | **atemp** | **hum** | **windspeed** | **casual** | **registered** |
| **0** | 1 | 0 | 1 | 0 | 0 | 6 | 0 | 1 | 0.24 | 0.2879 | 0.81 | 0.0 | 3 | 13 |
| **1** | 1 | 0 | 1 | 1 | 0 | 6 | 0 | 1 | 0.22 | 0.2727 | 0.80 | 0.0 | 8 | 32 |
| **2** | 1 | 0 | 1 | 2 | 0 | 6 | 0 | 1 | 0.22 | 0.2727 | 0.80 | 0.0 | 5 | 27 |
| **3** | 1 | 0 | 1 | 3 | 0 | 6 | 0 | 1 | 0.24 | 0.2879 | 0.75 | 0.0 | 3 | 10 |
| **4** | 1 | 0 | 1 | 4 | 0 | 6 | 0 | 1 | 0.24 | 0.2879 | 0.75 | 0.0 | 0 | 1 |

Out[11]:

In [12]:

y **=** df.iloc[:,**-**1:]

y.head()

Out[12]:

|  |  |
| --- | --- |
|  | **cnt** |
| **0** | 16 |
| **1** | 40 |
| **2** | 32 |
| **3** | 13 |
| **4** | 1 |

### Spliting data into test and training set

In [13]:

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(x, y, test\_size**=**0.30)

In [14]:

print("Dataset shape:", df.shape)

print("Input Features shape: ", X\_train.shape, y\_train.shape)

print("Output Features shape: ", X\_test.shape, y\_test.shape)

Dataset shape: (17379, 15)  
Input Features shape: (12165, 14) (12165, 1)  
Output Features shape: (5214, 14) (5214, 1)

### Applying Linear Regression

In [15]:

**from** sklearn.linear\_model **import** LinearRegression

lin **=** LinearRegression()

#### Fitting model

In [16]:

lin.fit(X\_train,y\_train)

Out[16]:

LinearRegression()

#### Predicting values

In [17]:

pred **=** lin.predict(X\_test)

In [18]:

pred

Out[18]:

array([[289.],  
 [461.],  
 [337.],  
 ...,  
 [487.],  
 [170.],  
 [108.]])

### Evaluation Matrix

In [19]:

**from** sklearn.metrics **import** r2\_score, mean\_squared\_error, mean\_absolute\_error

In [20]:

print('Mean Absolute Error:', mean\_absolute\_error(y\_test, pred))

print('Mean Squared Error:', mean\_squared\_error(y\_test, pred))

print('Root Mean Squared Error:', np.sqrt(mean\_squared\_error(y\_test, pred)))

print('R squared Error:', r2\_score(y\_test, pred))

Mean Absolute Error: 1.0648290841326865e-13  
Mean Squared Error: 2.06387408709926e-26  
Root Mean Squared Error: 1.436618977703991e-13  
R squared Error: 1.0

## Diabetes Dataset

### Objective

The objective of the dataset is to diagnostically predict whether or not a patient has diabetes

#### Import pandas, numpy, seaborn, matplotlib.pyplot packages

In [30]:

**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

**%**matplotlib inline

**import** seaborn **as** sns

**from** warnings **import** filterwarnings

filterwarnings('ignore')

In [31]:

df **=** pd.read\_csv('Datasets/diabetes.csv')

df.head()

Out[31]:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** | **Outcome** |
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 | 1 |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 | 0 |
| **2** | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 | 1 |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 | 0 |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 | 1 |

**It is a Classification Problem - the Dependent variable is Outcome**

* **Shape of Dataset**

In [32]:

df.shape

Out[32]:

(768, 9)

* **The dataset has total 768 rows & 9 Attributes**
* **Checking Information of Dataset**

In [4]:

df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 768 entries, 0 to 767  
Data columns (total 9 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Pregnancies 768 non-null int64   
 1 Glucose 768 non-null int64   
 2 BloodPressure 768 non-null int64   
 3 SkinThickness 768 non-null int64   
 4 Insulin 768 non-null int64   
 5 BMI 768 non-null float64  
 6 DiabetesPedigreeFunction 768 non-null float64  
 7 Age 768 non-null int64   
 8 Outcome 768 non-null int64   
dtypes: float64(2), int64(7)  
memory usage: 54.1 KB

* Dataset has 2 Float columns, 7 integer columns

## Data preprocessing

#### checking null values

In [5]:

df.isnull().sum()

Out[5]:

Pregnancies 0  
Glucose 0  
BloodPressure 0  
SkinThickness 0  
Insulin 0  
BMI 0  
DiabetesPedigreeFunction 0  
Age 0  
Outcome 0  
dtype: int64

* There is no null values

### Spliting data

In [33]:

x **=** df.drop(['Outcome'], axis**=**1)

x.head()

Out[33]:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** |
| **0** | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 |
| **1** | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 |
| **2** | 8 | 183 | 64 | 0 | 0 | 23.3 | 0.672 | 32 |
| **3** | 1 | 89 | 66 | 23 | 94 | 28.1 | 0.167 | 21 |
| **4** | 0 | 137 | 40 | 35 | 168 | 43.1 | 2.288 | 33 |

In [34]:

y **=** df.iloc[:,**-**1:]

y.head()

Out[34]:

|  |  |
| --- | --- |
|  | **Outcome** |
| **0** | 1 |
| **1** | 0 |
| **2** | 1 |
| **3** | 0 |
| **4** | 1 |

### By Standardising features

In [35]:

**from** sklearn.preprocessing **import** StandardScaler

sc **=** StandardScaler()

In [36]:

sc\_input **=** sc.fit\_transform(df.drop(['Outcome'], axis **=** 1))

sc\_input

Out[36]:

array([[ 0.63994726, 0.84832379, 0.14964075, ..., 0.20401277,  
 0.46849198, 1.4259954 ],  
 [-0.84488505, -1.12339636, -0.16054575, ..., -0.68442195,  
 -0.36506078, -0.19067191],  
 [ 1.23388019, 1.94372388, -0.26394125, ..., -1.10325546,  
 0.60439732, -0.10558415],  
 ...,  
 [ 0.3429808 , 0.00330087, 0.14964075, ..., -0.73518964,  
 -0.68519336, -0.27575966],  
 [-0.84488505, 0.1597866 , -0.47073225, ..., -0.24020459,  
 -0.37110101, 1.17073215],  
 [-0.84488505, -0.8730192 , 0.04624525, ..., -0.20212881,  
 -0.47378505, -0.87137393]])

In [37]:

df\_input **=** pd.DataFrame(sc\_input, columns**=** df.columns[:**-**1])

df\_input.head()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Pregnancies** | **Glucose** | **BloodPressure** | **SkinThickness** | **Insulin** | **BMI** | **DiabetesPedigreeFunction** | **Age** |
| **0** | 0.639947 | 0.848324 | 0.149641 | 0.907270 | -0.692891 | 0.204013 | 0.468492 | 1.425995 |
| **1** | -0.844885 | -1.123396 | -0.160546 | 0.530902 | -0.692891 | -0.684422 | -0.365061 | -0.190672 |
| **2** | 1.233880 | 1.943724 | -0.263941 | -1.288212 | -0.692891 | -1.103255 | 0.604397 | -0.105584 |
| **3** | -0.844885 | -0.998208 | -0.160546 | 0.154533 | 0.123302 | -0.494043 | -0.920763 | -1.041549 |
| **4** | -1.141852 | 0.504055 | -1.504687 | 0.907270 | 0.765836 | 1.409746 | 5.484909 | -0.020496 |

Out[37]:

### Spliting into training and test data

In [38]:

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(df\_input, df['Outcome'], test\_size**=**0.30)

In [39]:

print("Dataset shape:", df.shape)

print("Input Features shape: ", X\_train.shape, y\_train.shape)

print("Output Features shape: ", X\_test.shape, y\_test.shape)

Dataset shape: (768, 9)  
Input Features shape: (537, 8) (537,)  
Output Features shape: (231, 8) (231,)

## Applying KNN for K=1

In [40]:

**from** sklearn.neighbors **import** KNeighborsClassifier

knn **=** KNeighborsClassifier(n\_neighbors**=** 1)

#### Training and Predicting

In [42]:

knn.fit(X\_train,y\_train)

pred **=** knn.predict(X\_test)

**from** sklearn.metrics **import** classification\_report,confusion\_matrix

**from** sklearn.model\_selection **import** cross\_val\_score

print('\n Accuracy Score:', accuracy\_score(y\_test, pred))

print("\n Confusion Matrix: ")

print(confusion\_matrix(y\_test, pred))

print('\n Classification Report: ')

print(classification\_report(y\_test, pred))

Accuracy Score: 0.7142857142857143  
  
 Confusion Matrix:   
[[117 30]  
 [ 36 48]]  
  
 Classification Report:   
 precision recall f1-score support  
  
 0 0.76 0.80 0.78 147  
 1 0.62 0.57 0.59 84  
  
 accuracy 0.71 231  
 macro avg 0.69 0.68 0.69 231  
weighted avg 0.71 0.71 0.71 231

### Analysing Visually by Error rate

In [43]:

error\_rate **=** [ ]

**for** i **in** range(1,40):

k **=** KNeighborsClassifier(n\_neighbors**=**i)

score **=** cross\_val\_score(k, df\_input , df['Outcome'], cv**=**10)

error\_rate.append(1**-**score.mean())

In [44]:

plt.figure(figsize**=**(10,6))

plt.plot(range(1,40), error\_rate, color**=**'blue', linestyle**=**'dashed', marker**=**'o', markerfacecolor**=**'red', markersize**=**10)

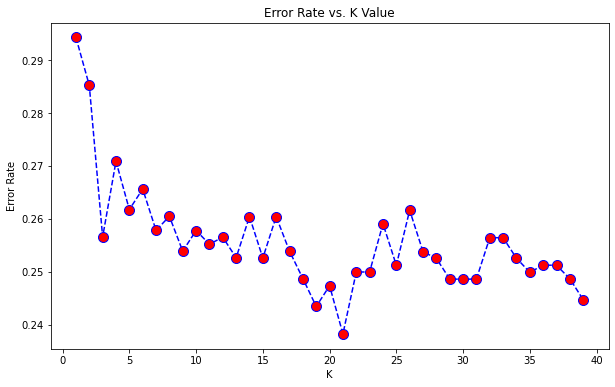
plt.title('Error Rate vs. K Value')

plt.xlabel('K')

plt.ylabel('Error Rate')

Out[44]:

Text(0, 0.5, 'Error Rate')



k > 27 error rate keeps decreasing. now finding clasification report at k=27

In [63]:

knn **=** KNeighborsClassifier(n\_neighbors**=**27)

knn.fit(X\_train,y\_train)

pred **=** knn.predict(X\_test)

print('WITH K=27')

print('\n Accuracy Score:', accuracy\_score(y\_test, pred))

print('\n Confusion Matrix:')

print(confusion\_matrix(y\_test,pred))

print('\n Classification Report:')

print(classification\_report(y\_test,pred))

WITH K=27  
  
 Accuracy Score: 0.7575757575757576  
  
 Confusion Matrix:  
[[135 12]  
 [ 44 40]]  
  
 Classification Report:  
 precision recall f1-score support  
  
 0 0.75 0.92 0.83 147  
 1 0.77 0.48 0.59 84  
  
 accuracy 0.76 231  
 macro avg 0.76 0.70 0.71 231  
weighted avg 0.76 0.76 0.74 231

In [64]:

fpr, tpr, threshold **=** metrics.roc\_curve(y\_test, pred)

roc\_auc **=** metrics.auc(fpr, tpr)

roc\_auc

Out[64]:

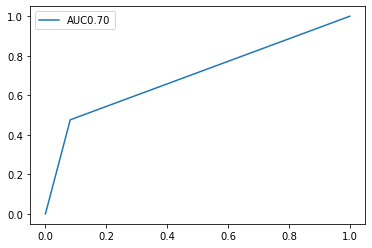
0.6972789115646258

In [65]:

plt.plot(fpr, tpr, label**=**'AUC%0.2f'**%**roc\_auc)

plt.legend()

plt.show()



### Applying Logistic Regression

In [53]:

**from** sklearn.linear\_model **import** LogisticRegression

clf **=** LogisticRegression()

### Fitting model

In [54]:

clf.fit(X\_train,y\_train)

Out[54]:

LogisticRegression()

### Predicting values

In [55]:

ypred **=** clf.predict(X\_test)

In [56]:

pred

Out[56]:

array([1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1,  
 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1,  
 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,  
 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,  
 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,  
 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0,  
 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0,  
 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1,  
 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,  
 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,  
 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1], dtype=int64)

### Accuracy of logistic regression

In [26]:

**from** sklearn.metrics **import** accuracy\_score

In [57]:

accuracy\_score(y\_test, ypred)

Out[57]:

0.7619047619047619

In [58]:

print(confusion\_matrix(y\_test, ypred))

[[124 23]  
 [ 32 52]]

In [59]:

print(classification\_report(y\_test, ypred))

precision recall f1-score support  
  
 0 0.79 0.84 0.82 147  
 1 0.69 0.62 0.65 84  
  
 accuracy 0.76 231  
 macro avg 0.74 0.73 0.74 231  
weighted avg 0.76 0.76 0.76 231

In [60]:

**from** sklearn **import** metrics

In [61]:

fpr, tpr, threshold **=** metrics.roc\_curve(y\_test, ypred)

roc\_auc **=** metrics.auc(fpr, tpr)

roc\_auc

Out[61]:

0.7312925170068029

In [62]:

plt.plot(fpr, tpr, label**=**'AUC%0.2f'**%**roc\_auc)

plt.legend()

plt.show()

